利用CUDA快速计算三角网格模型简化误差

张 衡,唐 杰,武港山
(南京大学 软件新技术国家重点实验室,江苏 南京 210093)

摘 要:提出从整体上考虑三角网格模型的简化误差的方法。该方法综合考虑了三角网格模型上所有采样点对误差的贡献,较以往方法只考虑最大误差更加准确。利用CUDA实现了高度并行化的网格模型简化误差计算算法。算法采用平均单元格来组织三角片,以便快速空间查询。同时设计了相应的数据结构,克服了CUDA没有指针、不能动态申请资源、尽量避免同步操作等问题。最后通过实验证明了本算法在速度和数据量上的优越性,并且阐述了GPU相对于CPU在高性能计算上的优点。

关键词:三角网格;CUDA;简化误差;平均单元格;并行
中图分类号:TP391 **文献标识码**:A **文章编号**:1673-629X(2011)07-0001-04

Fast Calculating Simplification Error of Triangular Mesh Using CUDA

ZHANG Heng, TANG Jie, WU Gang-shan
(National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210093, China)

Abstract: Propose a method to evaluate the overall simplification error of triangular meshes. It considers the contribution of all sample points to the error, which is more precise than the former methods that only consider the maximum error. Achieve a highly paralleled algorithm to calculate the simplification error of triangular meshes by using CUDA. Uniform gird is adopted to manage triangles of each mesh to accelerate the space search speed. Proper data structure is designed to solve the problems such as CUDA does not support pointer, dynamically memory allocating and to avoid synchronization as much as possible. Finally, experimental results show the efficiency and effectiveness of this algorithm, which verify the advantages of GPU on high performance calculation compared to CPU.

Key words: triangular mesh; CUDA; simplification error; uniform grid; parallel

0 引 言

网格模型的简化误差评估广泛应用于网格简化、网格多分辨分析、网格变形等应用中。目前的网格误差评估算法有不少。Cignoni[1]最早介绍了简化模型与原始模型之间的误差评估。而后Aspert[2]提出了另外一种更快速的方法,但是消耗更多内存。Garland[3]等提出了一种二次误差测度的网格误差评估机制,计算速度快,因而得到广泛的应用。但该算法的不足之处是其计算的是点到三角片所在平面的距离而不是点到三角片的距离,这会引入一些误差。

在评判简化误差采取的度量中,Hausdorff距离是较为常用的一种评估机制,但其计算量相当大。对于三维空间中的具有n个多边形的网格模型,已知最好算法精确计算Hausdorff距离的时间复杂度是$O(n^{(3+\varepsilon)})$,其中$\varepsilon > 0$[4],可见当n较大时,计算复杂度比较高。Guthe[5]提出了一种Hausdorff距离的快速算法,该算法能够快速定位模型距离较近的区域,减少了大量的无用计算。最近,Tang[6]提出了一种速度更快的方法,但该方法计算的是近似距离,且对模型要求较高,只能处理封闭模型。

上述方法在处理的模型数据量较大时,速度依旧不理想。此外,Hausdorff距离只考虑了简化模型与原始模型之间的最大误差。该误差对简化误差的表示并不够准确,如果需要在整体上考虑,则计算量更大,因此如何提高计算速度和误差评估的准确性已成为一个迫切需要解决的问题。

随着显卡的发展,GPU由于其高度的并行性,计算能力已经超越了通用的CPU,能够用来解决商业、工业以及科学方面的复杂计算问题,而CUDA[7]的出现更是降低了GPU编程方面的复杂度。CUDA一经

收稿日期:2010-11-30;**修回日期**:2011-03-05
基金项目:国家高技术研究发展计划(863)(2007AA06A402);江苏省自然科学基金(BK2008262)
作者简介:张 衡(1982-),男,无锡人,硕士研究生,研究方向为计算机图形学、三维建模;唐 杰,博士,副教授,CCF会员,研究方向为计算机图形学、三维建模。

推出,立刻成为研究热点,如 Satish[8] 利用 CUDA 实现多种并行排序算法,Zhou[9] 利用 GPU 实现了并行曲面重建等。

文中提出了一种基于 CUDA 的网格模型简化误差计算方法。该方法从整体上考虑模型的简化误差,避免了以往利用 Hausdorff 距离来评估简化误差时,只考虑了模型间的最大误差,而忽略了整体误差的不足。此外,该算法在 GPU 上利用 CUDA 实现了高度并行化的网格模型简化误差计算。算法针对 CUDA 没有指针、不能动态分配资源等限制,设计了有效的算法和数据结构,克服了上述问题。同现有的基于 CPU 的算法相比,本算法具有更快的运行速度,且计算出的误差能更好地反映简化网格与原始网格的差异。

1 网格模型简化误差

网格模型的简化误差是用来评估简化模型和原始模型误差的一个度量。目前常用对称 Hausdorff 距离来评估两个网格模型之间的几何误差,单向 Hausdorff 距离以及对称 Hausdorff 距离的数学定义如下[10]:

定义 1 三维空间中的一点 x 到网格模型 M 的距离 d_E 定义为:

$$d_E(x,M)=\min_{y\in M}(d(x,y)) \tag{1}$$

其中,$d(x,y)$ 是点 x 到点 y 的欧氏距离。

定义 2 网格 M 到 M' 的单向 Hausdorff 距离 d_s 定义为:

$$d_s(M,M')=\max_{x\in M}(d_E(x,M')) \tag{2}$$

定义 3 网格 M 到 M' 的对称 Hausdorff 距离 d_H 定义为:

$$d_H(M,M')=\max(d_s(M,M'),d_s(M',M)) \tag{3}$$

Hausdorff 距离考虑的仅仅是两个网格模型之间的最大误差,属于 L_∞ 误差。这在某种程度上非常容易受噪声的干扰,从而给出错误的评价。文中在传统以对称 Huasdorff 距离为基础的网格误差评判标准上,考虑所有采样点与相比较模型的误差,以它们的均方和作为新的简化误差评估标准,解决了传统方法的缺陷,提高了简化误差的准确性。

新的评估标准定义如下:

定义 4 网格 M 到 M' 的单向整体简化误差 d_{sw} 定义为:

$$d_{sw}(M,M')=\left(\sum_{x\in M} d_E^{\,2}(x,M')\right)/\mathrm{count}(M) \tag{4}$$

定义 5 网格 M 到 M' 的对称整体简化误差 d_{hw} 定义为:

$$d_{hw}(M,M')=\max(d_{sw}(M,M'),d_{sw}(M',M)) \tag{5}$$

2 算法概述

为计算 A,B 模型之间的简化误差,需要计算 A 所有采样点到 B 的所有三角片的最短距离,计算量相当大,可以采用平均单元格[11]减少三角片的比较次数。

2.1 CUDA 下三角网格模型的平均单元格划分

建立平均单元格,就是将三角片分配到某个单元格中,这样可以快速地排除大量不必要进行计算的三角形,提高计算速度。

目前 CPU 上的三角网格平均单元格划分算法很多,如 Lagae[12],但是在 CUDA 上创建平均单元格会遇到一些问题:

(1)每个三角片属于多个单元格,有三角片的单元格的个数也无法预先知道,这导致平均单元格所占内存大小难以确定;

(2)CUDA 不支持指针且不能动态分配资源。因此,现有 CPU 算法并不能直接用于 GPU。文中实现了一种适合 GPU 的空间平均单元格划分算法,Kalojanov[13] 也采用了类似的方法。

2.1.1 数据结构

由于 CUDA 无指针结构,所以一般采用索引来避免指针操作。数据存储方式都是数组,或者数组对。

如图 1 所示,grids 表示单元格索引,triangles 表示三角片索引。数组对 pairs,把 grids 和 triangles 的索引对应起来,表示哪些三角形在哪个单元格内,采用冗余存储方式,一个三角形可以属于多个单元格。cells 数组,用于记录 pairs 索引中每个 grid 的起始和终止位置。

图 1 所示为:grid1 中有三个三角片,分别是 t2、t4、t5。它们在 triangles 数组中的位置是第 4 个元素到第 6 个元素。

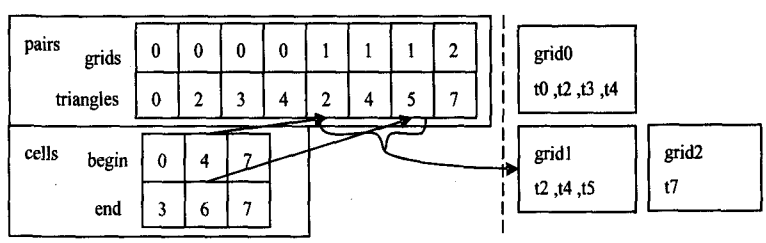


图 1 平均单元格划分算法数据结构

2.1.2 算法步骤

步骤 1:计算平均单元格分辨率,开辟 cells 数组对。平均单元格分辨率并不是越高越好,因为如果某些单元格没有点,但在遍历时仍然要经过这些单元格,这需要消耗时间,所以以模型的包围盒和点的个数为参考,尽量使得每个单元格里有 1 个点。

步骤 2:利用 CUDA 对每个三角片开辟线程,来计算它穿越的单元格数,并用总和大小来建立 pairs 数组对,建立 offset 数组记录每个三角片在 pairs 数组对中

的起始位置。

步骤3:利用CUDA对每个三角片开辟线程,计算它所在的平均单元格并且写入pairs数组,由于CUDA没有指针,又为了避免同步,利用offset数组可以避免写冲突,实现完全并行。

步骤4:利用CUDA对pairs数组对进行排序,使其从以triangles索引升序排列转变为以grids索引为升序排列的数组。

步骤5:利用CUDA对pairs数组对中的每个对开辟线程,比较这个对和它前面对之间的grids索引号,如果不同则判定为两个单元格在pairs数组对中的交界位置,填入cells数组对。

除了步骤1这个计算量很小的步骤,其他都用CUDA并行实现,速度相当快。

2.1.3 共享内存的应用

CUDA中的共享内存和全局内存相比,虽然访问受到要在一个块中的限制,但是速度相对全局内存有百倍左右的差别。上面的算法中多次用到了共享内存。

在步骤2中,如果把每个线程的三角片数都传到全局内存,然后在全局内存相加,相当费时间。所以文中利用共享内存,把每个块中的线程计算出来的三角片数存入共享内存。这个块的所有计算完成后在共享内存中,把它们加起来,然后再传回全局内存。最后只需要把所有块传出的数据加一下,这样大大减少了全局内存的访问次数。

在步骤5中,如果每个对都从全局内存提取自己对应元素和前一个元素比较的话,需要$2n$次(n为pairs数组对的长度)全局内存访问,现在采取每个线程都把自己对应元素读入共享内存,这样它的后一个元素就不需要再访问全局内存了,直接访问共享内存,但由于各块之间共享内存不能共享,所以每个块的第一个元素还是需要从全局内存读取,这样总的访问次数就是$n+m$(m为块个数)。m远小于n可以忽略,节约了近一半的时间。

2.2 CUDA下三角形网格模型的相似距离计算

在平均单元格数据结构的帮助下,在GPU上实现了一个高度并行的网格误差计算算法,该算法的CUDA程序伪代码如图2所示。

上述算法中,点到网格模型的距离由点到三角片距离中最小值决定。以点所在单元格为中心,不断扩大搜索范围,每扩大一层都在原来正方体外围加一层单元格。在计算过程中,用一个变量存储现在计算的点到将要扩大的那一层单元格区域的最短距离。如果在现在的搜索范围内已经找到点与某三角片的距离小于等于这个变量,这就表明采样点离现在搜索范围外面的三角片的距离必然小于等于这个距离。

算法中点到三角片的距离计算如下:

给定空间中一点v,以及一个三角片$v_0v_1v_2$,根据点v在该三角片所在平面上的投影的位置,v到该三角片的距离d_v定义为如下三种情况(如图3所示,图中的6条虚线分别垂直于三角形的三条边):

```
1: __global__ Hausdorff_kernel(){
2    :int vidx←blockIdx.x * blockDim.x+threadIdx.x;//计算该线程采
     样点的索引
3:   if(vidx>= sample_point_num) return;
4:   gridIdx←计算点 vidx 所在单元格的索引;
5:   设置当前单元格为搜索区域
6:   float hDis←FLOAT_MIN
7:   int flag←0;
8:   do{
9:     flag←0;
10:    hDis←点 vidx 到搜索区域内所有三角片的最近距离
11:    对于搜索区域的六个边界平面
12:    if(点 vidx 到该边界面的距离 < hDis){
13:          在该方向上的搜索区域扩大一个单元格的步长
14:          flag←1;
15:    }
16:  }while(flag)
17:}
```

图2　CUDA程序伪代码

当v的投影v'落在区域1中时,d_v为点v到该三角片所处平面的距离;到v距离最近的点为v在该平面上的投影;

当v的投影落在区域2中时,d_v为点v到相应边的距离;到v距离最近的点为v在该边上的投影;

当v的投影v'落在区域3中时,d_v为点v到相应顶点的距离;到v距离最近的点为相应顶点。

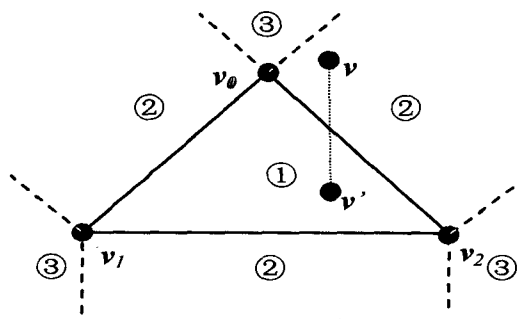


图3　点在三角形上的投影

3 实验及性能分析

本实验的实验环境采用具有4GB的显存,240个计算内核的Nvidia Tesla C1060。实验模型来自斯坦福大学3D数据库[14]中的Bunny模型和Armadillo模型并生成了它们的简化模型。

表1给出了模型参数及实验结果。

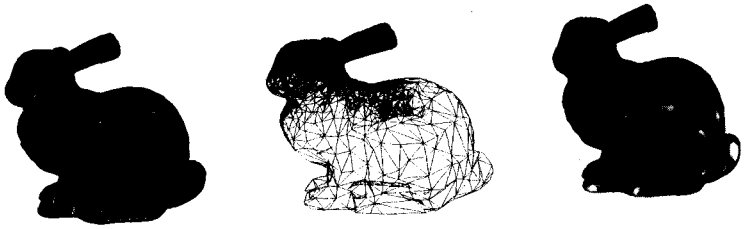
实验效果如图4所示。

表2所示为Tang[6]和Guthe[5]等人的实验环境、实验结果以及与文中结果的比较。

虽然由于实验环境不同,但是可以从速度提升的幅度上看出GPU的美好前景。提速的主要原因是GPU与CPU不同,GPU计算内核数大大多于CPU,GPU线程切换为硬件切换,耗时非常少,所以GPU非常适合逻辑简单的大规模数据并行处理。如若数据量更大,又采用内核较多的GPU,则本算法的速度优势越能表现出来。

表1 模型参数及实验结果

模型	Bunny	Bunny_Low	Armadillo	Armadillo_Low
顶点数	35947	13030	171550	6765
三角形数	69451	26069	343096	13188
空间单元格构建时间(ms)	7.7		16.8	
双向相似距离计算时间(ms)	39.6		106	
总时间(ms)	47		123	



原始模型 简化模型 简化误差用颜色表示(黑色最小,白色最大)

图4 实验效果图

表2 与Tang和Guthe在bunny模型计算速度上的比较

算法	运行环境	计算时间(ms)
Guthe	AMD Athlon 3000+(CPU)	6800
Tang	Intel 2.6GHz(CPU)	948
Our	Nvidia Tesla C1060(GPU)	47

4 结束语

文中提出了一种基于CUDA的三角网格模型简化误差的快速计算算法。算法利用GPU计算的高并行性,从总体上考虑了模型上所有采样点对简化误差的影响,比以往利用Hausdorff距离的表示更加准确。同时,文中在CUDA上实现了高度并行的网格模型简化误差评估算法,大幅提升了运算速度。最后通过实验证明了本算法的有效性。

GPU的通用计算是目前的研究热点。然而,GPU计算还存在一些问题,如现有很多算法的并行性不易挖掘,GPU的规格较多等。如何开发设计具备高度适应性的并行算法是下一步要考虑的问题。

参考文献:

[1] Cignoni P, Rocchini C, Scopigno R. Metro: Measuring error on simplified surfaces[J]. Computer Graphics Forum, 1998, 17(2): 167-174.

[2] Aspert N, Santa-Cruz D, Ebrahimi T. MESH: Measuring errors between surfaces using the Hausdorff distance[C]//Proceedings of the IEEE International Conference on Multimedia and Expo. Lausanne, Switzerland: [s. n.], 2002: 705-708.

[3] Garland M, Paul S H. Surface simplification using quadric error metrics[C]//Computer Graphics Proceedings, Annual Conference Series, ACM SIGGRAPH. Los Angeles, California: [s. n.], 1997: 209-216.

[4] Alt H, Brass B, Godau M, et al. Computing the Hausdorff distance of geometric patterns and shapes[J]. Discrete and Computational Geometry, 2003, 25: 65-76.

[5] Guthe M, Borodin P, Klein R. Fast and accurate Hausdorff distance calculation between meshes[C]//International Conferences in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG). [s. l.]: [s. n.], 2005: 41-48.

[6] Tang Min, Leey M, Kimz Y J. Interactive Hausdorff distance computation for general polygonal models[J]. ACM Transactions on Graphics, 2009, 28(3): 1-9.

[7] NVIDIA CUDAC programming guide 3.1[EB/OL]. 2010-05-28[2010-07-06]. http://developer.nvidia.com/object/cuda_3_1_downloads.html.

[8] Satish N, Harris M, Garland M. Designing efficient sorting algorithms for manycore GPUs[C]//Proceedings of the 2009 IEEE International Symposium on Parallel & Distributed Processing. Washington DC. USA: IEEE Computer Society, 2009: 1-10.

[9] Zhou Kun, Gong Minmin, Huang Xin, et al. Highly Parallel Surface Reconstruction[R]. [s. l.]: Microsoft Research Asia, 2008.

[10] 唐 杰, 张福炎. 任意网格模型的相似度评估[J]. 系统仿真学报, 2005, 17(1): 16-24.

[11] Fujimoto A, Tanaka T, Lwata K. ARTS: Accelerated ray-tracing system[C]//IEEE Computer Graphics and Applications. Los Alamitos, CA, USA: [s. n.], 1986: 16-26.

[12] Lagae A, Dutr P. Compact fast and robust grids for ray tracing[C]//Proceedings of the 19th Eurographics Symposium on Rendering. [s. l.]: [s. n.], 2008: 1235-1244.

[13] Kalojanov J, Slusallek P. A parallel algorithm for construction of uniform grids[C]//Proceedings of the Conference on High Performance Graphics. [s. l.]: [s. n.], 2009: 23-28.

[14] The Stanford 3D Scanning Repository[EB/OL]. 2010-03-03[2010-07-06]. http://graphics.stanford.edu/data Dscanrep.